Software Evaluation: Criteria-based Assessment

Mike Jackson, Steve Crouch and Rob Baxter

Criteria-based assessment is a quantitative assessment of the software in terms of sustainability, maintainability, and usability. This can inform high-level decisions on specific areas for software improvement.

A criteria-based assessment gives a measurement of quality in a number of areas. These areas are derived from *ISO/IEC 9126-1 Software engineering — Product quality*[1] and include usability, sustainability and maintainability.

The assessment involves checking whether the software, and the project that develops it, conforms to various characteristics or exhibits various qualities that are expected of sustainable software. The more characteristics that are satisfied, the more sustainable the software. Please note that not all qualities have equal weight e.g. having an OSI-approved open source licence is of more importance than avoiding TAB characters in text files.

In performing the evaluation, you may want to consider how different user classes affect the importance of the criteria. For example, for Usability-Understandability, a small set of well-defined, accurate, task-oriented user documentation may be comprehensive for Users but inadequate for Developers. Assessments specific to user classes allow the requirements of these specific user classes to be factored in and so, for example, show that a project rates highly for Users but poorly for Developers, or vice versa.

Scoring can also be affected by the nature of the software itself e.g. for Learnability one could envisage an application that has been well-designed, offers context-sensitive help etc. and consequently is so easy to use that tutorials aren't needed. Portability can apply to both the software and its development infrastructure e.g. the open source software OGSA-DAI[2] can be built, compiled and tested on Unix, Windows or Linux (and so is highly portable for Users and User-Developers). However, its Ruby test framework cannot yet run on Windows, so running integration tests would involve the manual setup of OGSA-DAI servers (so this is far less portable for Developers and, especially, Members).

The assessment criteria are grouped as follows.

Criterion	Sub-criterion	Notes – to what extent is/does the software…
Usability	Understandability	Easily understood?
	Documentation	Comprehensive, appropriate, well-structured user documentation?
	Buildability	Straightforward to build on a supported system?
	Installability	Straightforward to install on a supported system?
	Learnability	Easy to learn how to use its functions?
Sustainability and maintainability	Identity	Project/software identity is clear and unique?
	Copyright	Easy to see who owns the project/software?
	Licencing	Adoption of appropriate licence?

[1] http://www.iso.org/

[2] http://sourceforge.net/projects/ogsa-dai

	Governance	Easy to understand how the project is run and the development of the software managed?
	Community	Evidence of current/future community?
	Accessibility	Evidence of current/future ability to download?
	Testability	Easy to test correctness of source code?
	Portability	Usable on multiple platforms?
	Supportability	Evidence of current/future developer support?
	Analysability	Easy to understand at the source level?
	Changeability	Easy to modify and contribute changes to developers?
	Evolvability	Evidence of current/future development?
	Interoperability	Interoperable with other required/related software?

The rest of this document covers each category in greater depth, with lists of questions that we use at the Software Sustainability Institute when compiling detailed software evaluation reports.

Usability

Understandability How straightforward is it to understand: • What the software does and its purpose? • The intended market and users of the software? • The software's basic functions? • The software's advanced functions?	**Yes/No, supporting comments if warranted**
High-level description of what/who the software is for is available.	
High-level description of what the software does is available.	
High-level description of how the software works is available.	
Design rationale is available – why it does it the way it does.	
Architectural overview, with diagrams, is available.	
Descriptions of intended use cases are available.	
Case studies of use are available.	

Documentation Looking at the user documentation, what is its • Quality? • Completeness? • Accuracy? • Appropriateness? • Clarity?	**Yes/No, supporting comments if warranted**
Provides a high-level overview of the software.	
Partitioned into sections for users, user-developers and developers (depending on the software).	
States assumed background and expertise of the reader, for each class of user.	
Lists resources for further information.	
Further information is suitable for the level of the reader, for each class of user.	
Is task-oriented.	
Consists of clear, step-by-step instructions.	
Gives examples of what the user can see at each step e.g. screen shots or command-line excerpts.	
For problems and error messages, the symptoms and step-by-step solutions are provided.	
Does not use terms like "intuitive", "user friendly", "easy to use", "simple" or "obviously", unless as	

part of quotes from satisfied users	
States command names and syntax, says what menus to use, lists parameters and error messages exactly as they appear or should be typed.	
Uses `teletype-style fonts` for command-line inputs and outputs, source code fragments, function names, class names etc.	
For Java, the package names of classes are stated the first time a class is mentioned.	
English language descriptions of commands or errors are provided but only to complement the above.	
Plain-text files (e.g. READMEs) use indentation and underlining (e.g. === and ---) to structure the text.	
Plain-text files (e.g. READMEs) do not use TAB characters to indent the text.	
API documentation e.g. JavaDoc or Doxygen, documents APIs completely e.g. configuration files, property names etc.	
Is held under version control alongside the code.	
Is on the project web site.	
Documentation on the project web site makes it clear what version of the software the documentation applies to.	

Buildability How straightforward is it to: • Meet the pre-requisites for building the software on a build platform? • Build the software on a build platform?	**Yes/No, supporting comments if warranted**
Web site has instructions for building the software.	
Source distributions have instructions for building the software.	
An automated build (e.g. Make, ANT, custom solution) is used to build the software.	
Web site lists all third-party dependencies that are not bundled, along with web addresses, suitable versions, licences and whether these are mandatory or optional.	
Source distributions list all third-party dependencies that are not bundled, along with web addresses, suitable versions, licences and whether these are mandatory or optional.	
Dependency management is used to automatically download dependencies (e.g. ANT,	

Ivy, Maven or custom solution).	
All mandatory third-party dependencies are currently available.	
All optional third-party dependencies are currently available.	
Tests are provided to verify the build has succeeded.	

Installability How straightforward is it to: • Meet the pre-requisites for the software on a target platform? • Install the software onto a target platform? • Configure the software following installation for use? • Verify the installation for use? Note that in some cases build and install may be one and the same.	**Yes/No, supporting comments if warranted**
Web site has instructions for installing the software.	
Binary distributions have instructions for installing the software.	
Web site lists all third-party dependencies that are not bundled, along with web addresses, suitable versions, licences and whether these are mandatory or optional.	
Binary distributions list all third-party dependencies that are not bundled, along with web addresses, suitable versions, licences and whether these are mandatory or optional.	
Dependency management is used to automatically download dependencies (e.g. ANT, Ivy, Maven or custom solution).	
All mandatory third-party dependencies are currently available.	
All optional third-party dependencies are currently available.	
Tests are provided to verify the install has succeeded.	
When an archive (e.g. TAR.GZ or ZIP) is unpacked, it creates a single directory with the files within. It does not spread its contents all over the current directory.	
When software is installed, its contents are organised into sub-directories (e.g. docs for documentation, libs for dependent libraries) as	

appropriate.	
All source and binary distributions contain a README.TXT with project name, web site, how/where to get help, version, date, licence and copyright (or where to find this information), location of entry point into user doc.	
All GUIs contain a Help menu with commands to see the project name, web site, how/where to get help, version, date, licence and copyright (or where to find this information), location of entry point into user doc.	
All other content distributed as an archive contains a README.TXT with project name, web site, nature, how /where to get help, date.	
Installers allow user to select where to install software.	
Uninstallers uninstall every file or warns user of any files that were not removed and where these are.	

Learnability How straightforward is it to learn how to achieve: • Basic functional tasks? • Advanced functional tasks?	**Yes/No, supporting comments if warranted**
A getting started guide is provided outlining a basic example of using the software.	
Instructions are provided for many basic use cases.	
Instructions are provided supporting all use cases.	
Reference guides are provided for all command-line, GUI and configuration options.	
API documentation is provided for user-developers and developers.	

Sustainability and maintainability

Identity To what extent is the identity of the project/software clear and unique both within its application domain and generally?	**Yes/No, supporting comments if warranted**
Project/software has its own domain name.	
Project/software has a logo.	
Project/software has a distinct name within its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.	
Project/software has a distinct name regardless of its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.	
Project/software name does not throw up embarrassing "did you mean…" hits on Google.	
Project/software name does not violate an existing trade-mark.	
Project/software name is trade-marked.	

Copyright To what extent is it clear who wrote the software and owns its copyright?	**Yes/No, supporting comments if warranted**
Web site states copyright.	
Web site states who developed/develops the software, funders etc.	
If there are multiple web sites then these all state exactly the same copyright, licencing and authorship.	
Each source code file has a copyright statement.	
If supported by the language, each source code file has a copyright statement embedded within a constant.	
Each source code file has a licence header.	

Licencing Has an appropriate licence been adopted?	**Yes/No, supporting comments if warranted**
Web site states licence.	
Software (source and binaries) has a licence.	
Software has an open source licence.	

Software has an Open Software Initiative[3] (OSI)-recognised licence.	

Governance To what extent does the project make its management, or how its software development is managed, transparent?	**Yes/No, supporting comments if warranted**
Project has defined a governance policy.	
Governance policy is publicly available.	

Community To what extent does/will an active user community exist for this product?	**Yes/No, supporting comments if warranted**
Web site has statement of number of users/developers/members.	
Web site has success stories.	
Web site has quotes from satisfied users.	
Web site has list of important partners or collaborators.	
Web site has list of the project's publications.	
Web site has list of third-party publications that cite the software.	
Web site has list of software that uses/bundles this software.	
Users are requested to cite the project if publishing papers based on results derived from the software.	
Users are required to cite a boilerplate citation if publishing papers based on results derived from the software.	
Users exist who are not members of the project.	
Developers exist who are not members of the project.	

Accessibility To what extent is the software accessible?	**Yes/No, supporting comments if warranted**
Binary distributions are available (whether for free, payment, registration).	
Binary distributions are freely available.	
Binary distributions are available without the need for any registration or authorisation of	

[3] http://www.opensource.org/

access by the project.	
Source distributions are available (whether for free, payment, registration).	
Source distributions are freely available.	
Source distributions are available without the need for any registration or authorisation of access by the project.	
Access to source code repository is available (whether for free, payment, registration).	
Anonymous read-only access to source code repository.	
Ability to browse source code repository online.	
Repository is hosted externally to a single organisation/institution in a sustainable third-party repository (e.g. SourceForge, GoogleCode, LaunchPad, GitHub) which will live beyond the lifetime of any current funding line.	
Downloads page shows evidence of regular releases (e.g. six monthly, bi-weekly, etc.).	

Testability How straightforward is it to test the software to verify modifications?	**Yes/No, supporting comments if warranted**
Project has unit tests.	
Project has integration tests.	
For GUIs, project uses automated GUI test frameworks.	
Project has scripts for testing scenarios that have not been automated (e.g. for testing GUIs).	
Project recommends tools to check conformance to coding standards.	
Project has automated tests to check conformance to coding standards.	
Project recommends tools to check test coverage.	
Project has automated tests to check test coverage.	
A minimum test coverage level that must be met has been defined.	
There is an automated test for this minimum test coverage level.	
Tests are automatically run nightly.	
Continuous integration is supported – tests are automatically run whenever the source code	

changes.	
Test results are visible to all developers/members.	
Test results are visible publicly.	
Test results are e-mailed to a mailing list.	
This e-mailing list can be subscribed to by anyone.	
Project specifies how to set up external resources e.g. FTP servers, databases for tests.	
Tests create their own files, database tables etc.	

Portability To what extent can the software be used on other platforms?	**Yes/No, supporting comments if warranted**
Application can be built on and run under Windows.	
Application can be built on and run under Windows 7.	
Application can be built on and run under Windows XP.	
Application can be built on and run under Windows Vista.	
Application can be built on and run under UNIX/Linux.	
Application can be built on and run under Solaris.	
Application can be built on and run under RedHat.	
Application can be built on and run under Debian.	
Application can be built on and run under Fedora.	
Application can be built on and run under Ubuntu.	
Application can be built on and run under MacOSX.	
Browser applications run under Internet Explorer.	
Browser applications run under Mozilla Firefox.	
Browser applications run under Google Chrome.	
Browser applications run under Opera.	
Browser applications run under Safari.	

Supportability To what extent will the product be supported currently and in the future?	**Yes/No, supporting comments if warranted**
Web site has page describing how to get support.	
User doc has page describing how to get support.	
Software describes how to get support (in a	

README for command-line tools or a Help=>About window in a GUI).	
Above pages/windows/files describe, or link to, a description of "how to ask for help" e.g. cite version number, send transcript, error logs etc.	
Project has an e-mail address.	
Project e-mail address has project domain name.	
E-mails are read by more than one person.	
E-mails are archived.	
E-mail archives are publicly readable.	
E-mail archives are searchable.	
Project has a ticketing system.	
Ticketing system is publicly readable.	
Ticketing system is searchable.	
Web site has site map or index.	
Web site has search facility.	
Project resources are hosted externally to a single organisation/institution in a sustainable third-party repository (e.g. SourceForge, GoogleCode, LaunchPad, GitHub) which will live beyond the lifetime of the current project.	
E-mail archives or ticketing system shows that queries are responded to within a week (not necessarily fixed, but at least looked at and a decision taken as to their priority).	
If there is a blog, is it is regularly used.	
E-mail lists or forums, if present, have regular posts.	

Analysability How straightforward is it to analyse the software's source release to: • To understand its implementation architecture? • To understand individual source code files and how they fit into the implementation architecture?	**Yes/No, supporting comments if warranted**
Source code is structured into modules or packages.	
Source code structure relates clearly to the architecture or design.	
Project files for IDEs are provided.	
Source code repository is a revision control system.	

Structure of the source code repository and how this maps to the software's components is documented.	
Source releases are snapshots of the repository.	
Source code is commented.	
Source code comments are written in an API document generation mark-up language e.g. JavaDoc or Doxygen.	
Source code is laid out and indented well.	
Source code uses sensible class, package and variable names.	
There are no old source code files that should be handled by version control e.g. "SomeComponentOld.java".	
There is no commented out code.	
There are no TODOs in the code.	
Auto-generated source code is in separate directories from other source code.	
How to regenerate the auto-generated source code is documented.	
Coding standards are recommended by the project.	
Coding standards are required to be observed.	
Project-specific coding standards are consistent with community or generic coding standards (e.g. for C, Java, FORTRAN etc.).	

Changeability How straightforward is it to modify the software to: • Address issues? • Modify functionality? • Add new functionality?	**Yes/No, supporting comments if warranted**
Project has defined a contributions policy.	
Contributions policy is publicly available.	
Contributors retain copyright/IP of their contributions.	
Users, user-developers and developers who are not project members can contribute.	
Project has defined a stability/deprecation policy for components, APIs etc.	
Stability/deprecation policy is publicly available.	
Releases document deprecated components/APIs in that release.	
Releases document removed/changed	

components/APIs in that release.	
Changes in the source code repository are e-mailed to a mailing list.	
This e-mailing list can be subscribed to by anyone.	

Evolvability To what extent will the product be developed in the future: • For a future release? • Within a roadmap for the product?	**Yes/No, supporting comments if warranted**
Web site describes project roadmap or plans or milestones (either on a web page or within a ticketing system).	
Web site describes how project is funded/sustained.	
Web site describes end dates of current funding lines.	

Interoperability To what extent does the software's interoperability: • Meet appropriate open standards? • Function with required third-party components? • Function with optional third-party components?	**Yes/No, supporting comments if warranted**
Uses open standards.	
Uses mature, ratified, non-draft open standards.	
Provides tests demonstrating compliance to open standards.	